Unix File System Calls

BITS Pilani
Pilani Campus

Department of Computer Science and Information Systems

Points to be Covered

- File System Calls (Contd.)
  - dup
  - dup2
  - chmod
  - chown
  - link
  - unlink

dup() system call

The dup() system call creates a copy of the file descriptor oldfd, using the lowest-numbered unused file descriptor for the new descriptor.

Syntax: int dup(int oldfd);

Return Value: On success, these system calls return the new file descriptor. On error, -1 is returned.

dup() system call

```
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main()
{

   int file_desc = open("dup.txt", O_WRONLY | O_APPEND);
    if(file_desc < 0)
       printf("Error opening the file\n");

   int copy_desc = dup(file_desc);

   write(copy_desc,"This will be output to the file named dup.txt\n", 46);
   write(file_desc,"This will also be output to the file named dup.txt\n", 51);
    return 0;
}
```

dup2() system call

The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd. If the file descriptor newfd was previously open, it is silently closed before being reused.

Syntax: int dup2(int oldfd, int newfd);

Return Value: On success, these system calls return the new file descriptor. On error, -1 is returned.

dup2() system call

```
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>

int main()
{
   int file_desc = open("tricky.txt",O_WRONLY | O_APPEND);

   dup2(file_desc, 1) ;

   printf("I will be printed in the file tricky.txt\n");

   return 0;
}
```

dup() and dup2() system call

```
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>

int main()
{
    int fd1, fd2;
    fd1 = open("txt1.txt", O_RDONLY | O_CREAT, 777);
    close(1);   /* close standard output */
    dup(fd1);
}
```

dup() and dup2() system call

```
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>

int main()
{
    int fd1, fd2, fd3;
    fd1 = open("txt1.txt", O_RDONLY | O_CREAT, 777); //3
    fd2 = open("txt2.txt", O_RDONLY | O_CREAT, 777); //4

    //close (fd2);
    fd3=dup2(fd1, fd2);
}
```

chmod system call

- The **chmod** command is used to change the access mode of a file.
- Syntax: chmod [reference][operator][mode] filename
- The references are used to distinguish the users to whom the permissions apply i.e. they are list of letters that specifies whom to give permissions.

Reference	Class
u	owner
g	group
o	others
a	all

chmod system call

The operator is used to specify how the modes of a file should be adjusted. The following operators are accepted:

Operator	Description
+	Adds the specified modes to the specified classes
-	Removes the specified modes from the specified classes
=	The modes specified are to be made the exact modes for the specified classes

chmod system call

- The modes indicate which permissions are to be granted or removed from the specified classes. There are three basic modes which correspond to the basic permissions:

Mode	Description
r	Permission to read the file
w	Permission to write (or delete) the file
X	Permission to execute the file, or, in the case of a directory, search it.

chmod system call

- rw- rw- r-- CSIS CSIS text1.c
- rw- rw- r-- CSIS CSIS text2.c
- rwx rwx r-x CSIS CSIS BITS
- rw- rw- r-- CSIS CSIS semaphore.c
- rwx r-x r-x CSIS CSIS xyz.c

BEFORE: - rw- rw- r-- CSIS CSIS text1.c

COMMAND: chmod u=r text1.c

AFTER: - r-- rw- r-- CSIS CSIS text1.c

chmod system call

Let's restrict the permission such that the user cannot search the directory BITS.

- rw- rw- r-- CSIS CSIS text1.c
- rw- rw- r-- CSIS CSIS text2.c
- rwx rwx r-x CSIS CSIS BITS
- rw- rw- r-- CSIS CSIS semaphore.c
- rwx r-x r-x CSIS CSIS xyz.c

BEFORE: drwxrwxr-x CSIS CSIS BITS

COMMAND: chmod u=rw BITS
or: chmod u-1 BITS

AFTER: drw-rwxr-x CSIS CSIS BITS

chown system call

- The **chown** command changes the user and/or group ownership of a given file..

- Syntax
  - chown owner-user file
  - chown owner-user:owner-group file
  - chown owner-user:owner-group directory

chown system call

First, list permissions for demo.txt, enter:

ls -l demo.txt

Sample outputs: -rw-r--r-- 1 root root 0 Aug 31 05:48 demo.txt

In this example change file ownership to vivek user and list the permissions, run:

- chown vivek demo.txt

ls -l demo.txt

Sample outputs:

- -rw-r--r-- 1 vivek root 0 Aug 31 05:48 demo.txt

chown system call

Now, the owner is set to vivek followed by a colon and a group onwership is also set to vivek group, run:

chown vivek:vivek demo.txt

ls -l demo.txt

Sample outputs:-rw-r--r-- 1 vivek vivek 0 Aug 31 05:48 demo.txt

chown system call

Here, we have changed only the group of file. To do so, the colon and following GROUP-name ftp are given, but the owner is omitted, only the group of the files is changed:

chown :ftp demo.txt

ls -l demo.txt

Sample outputs:

-rw-r--r-- 1 vivek ftp 0 Aug 31 05:48 demo.txt

link() system call

link - make a new name for a file

Syntax: **int link(const char **oldpath*, const char **newpath*)**

- **link**() creates a new link (also known as a hard link) to an existing file.
- If *newpath* exists it will *not* be overwritten.
- This new name may be used exactly as the old one for any operation; both names refer to the same file (and so have the same permissions and ownership) and it is impossible to tell which name was the `original'.

Return Value: On success, zero is returned. On error, -1 is returned.

link() system call

- link ("/usr/src/uts/sys", "/usr/include/sys");
- link ("/usr/include/realfile.h", "/usr/src/uts/sys/testfile.h");

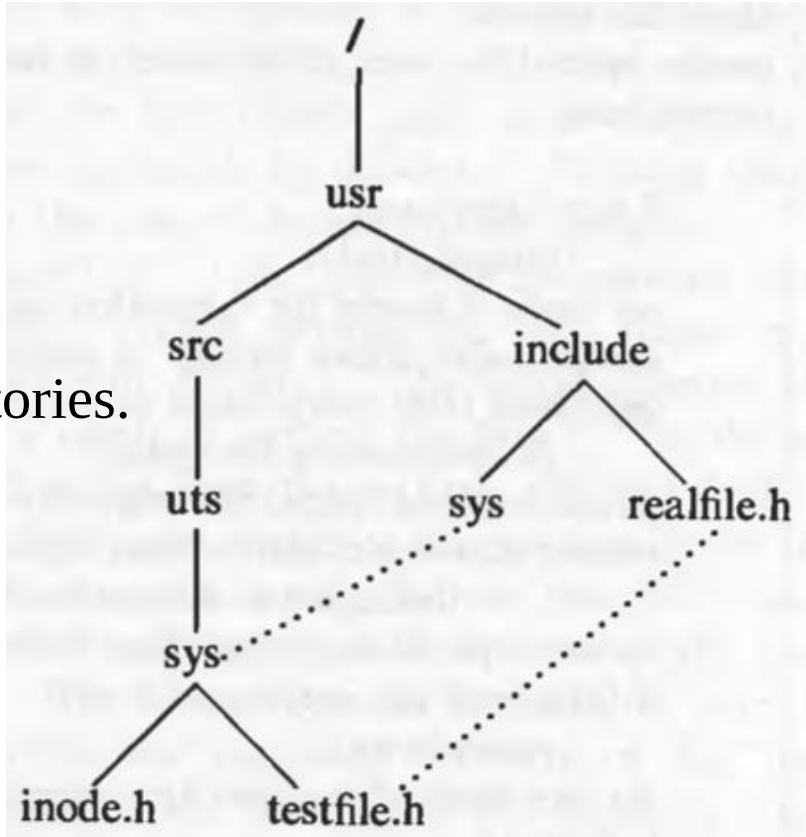
- 3 pathnames refer to the same file:

"/usr/src/uts/sys/testfile.h",

"/usr/include/sys/testfile.h",

"/usr/include/realfile"

- Only a superuser is allowed to link directories.



unlink() system call

unlink - delete a name and possibly the file it refers to

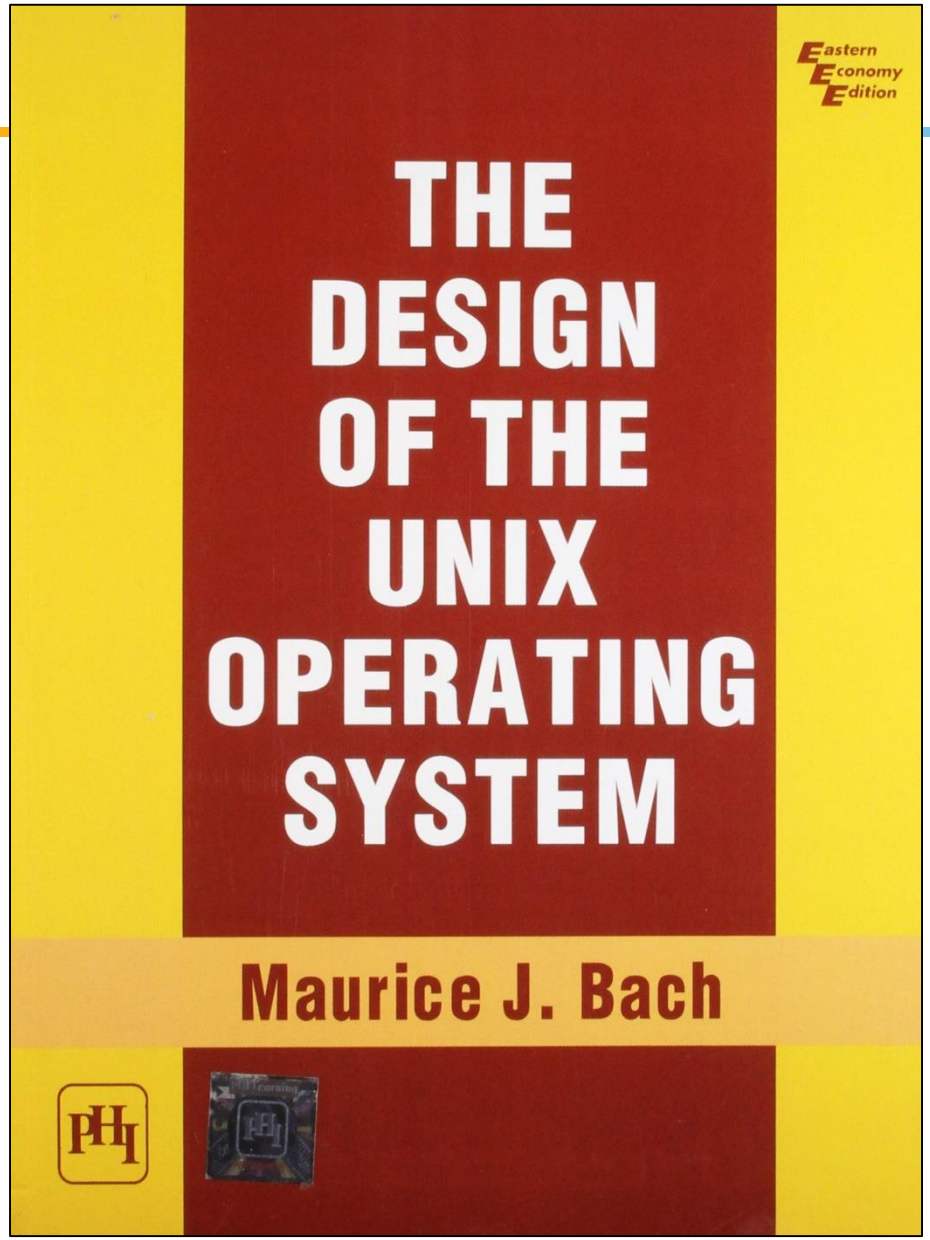
Syntax: **int unlink(const char **pathname*);**

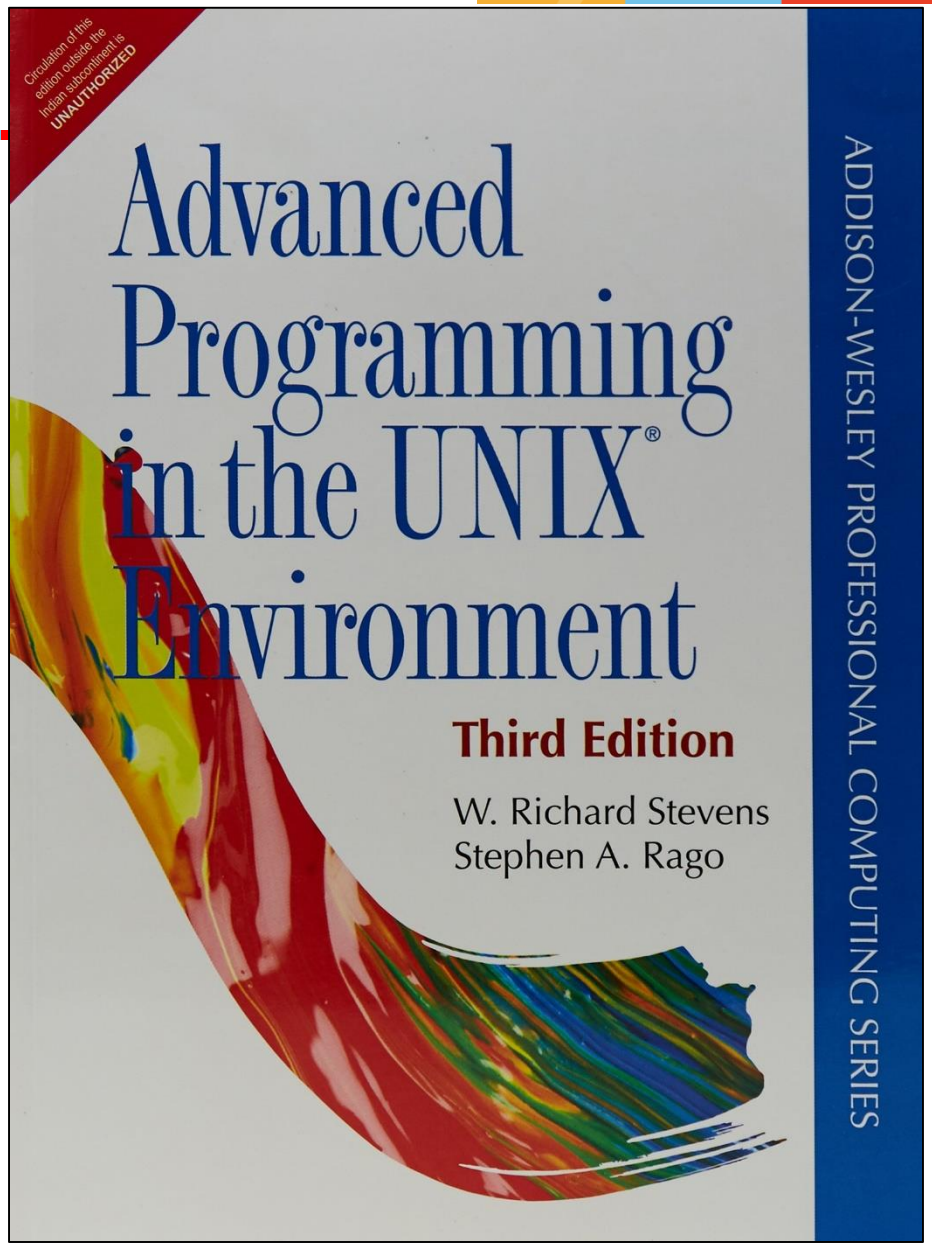
Return Value: On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

- unlink() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open, the file is deleted and the space it was using is made available for reuse.
- If the name was the last link to a file but any processes still have the file open, the file will remain in existence until the last file descriptor referring to it is closed.

Problem 1

Assume that the text file is already created. You need to open the file using O_RDWR flag only and print “hi All” from the program without use any printf or cout function.


Eastern Economy Edition
THE DESIGN OF THE UNIX OPERATING SYSTEM
Maurice J. Bach
PHI


Circulation of this edition outside the Indian subcontinent is UNAUTHORIZED
Advanced Programming in the UNIX® Environment
Third Edition
W. Richard Stevens
Stephen A. Rago
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

THE LINUX PROGRAMMING INTERFACE

A Linux and UNIX® System Programming Handbook

MICHAEL KERRISK

no starch press

Any Queries?